Integrating Automatic Verification of Safety Requirements in Railway Interlocking System Design

Giovanni Dipoppa[1], Giovanni D'Alessandro[2], Roberto Semprini[3], Enrico Tronci[4]

Abstract

A Railway Interlocking System *(RIS) is an Embedded System (namely a Supervisory Control System) that ensures the safe operation of the devices in a Railway Station. Of course a RIS is a* Safety Critical System

In this paper we explore the possibility of integrating automatic formal verification methods in a given industry RIS design flow.

The main obstructions to be overcome in our work are: selecting a formal verification tool that is efficient enough to solve the verification problems at hand and devising a cost effective integration strategy for such tool.

Eventually we were able to devise a successful integration strategy meeting the above constraints. This is done without requiring major modification in the preexistent design flow nor retraining of personnel.

We run verification experiments for a RIS designed for the Singapore Subway. Such experiments show that the RIS design flow obtained from our integration strategy will indeed be able to automatically verify real life *RIS designs.*

1 Introduction

A *Railway Interlocking System* (RIS) is an Embedded System (namely a Supervisory Control System) that ensures the safe operation of the devices in a Railway Station. E.g. a RIS ensures that it is not possible to set (manually or automatically by some other system) switch points in a way that may lead to train collision. Fig. 1 shows the role of an interlocking system in the railway control hierarchy.

It is quite obvious that a RIS is a *Safety Critical System*. Indeed RIS customers (typically Railway Companies) pay more and more attention to safety evidence of newly designed RISs. Certification authorities and upcoming standards (e.g. CENELEC EN50128/129 in Europe [4, 5]) also require stronger and stronger evidences arguing for the safety of each newly designed RIS.

Thus when designing a RIS much effort goes into ensuring its correctness w.r.t. given specifications. Needless to say this tends to raise production costs as well as time to market. This is worsened by the always increasing RIS complexity.

In this situation it is quite natural to explore methods to increase confidence in RIS design correctness (w.r.t. given specifications) and possibly decrease production times and costs. Many methods have been studied to address the above issues. For example see [6, 9, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26].

The goal of our work here is to effectively integrate automatic verification of a RIS implementation within a given industrial design flow. Examples of papers close to our are: [6, 7]. Note however that here we present a case study about *integration* of automatic verification in a given industrial design flow rather than a case study about *verification* of a given railway station (e.g. as in [7]).

Since we are interested in automatic verification we focus on RIS verification via model checking. Indeed, since RISs are finite state machines, looking at automatic verification via model checking is quite natural.

Automatic verification via *Model Checking* [2] has been very successful for hardware design. In fact it

[1]ENEA C.R.-CASACCIA, TISGI Section, Via Anguillarese, 301, S.Maria di Galeria I-00060 ROMA - ITALY,
email: `giovanni.dipoppa@casaccia.enea.it`
url: `http://tisgi.casaccia.enea.it`
This research has been partially supported by ESPRIT project ISA-EUNET EP27450

[2]CASPUR, c/o CICS Universita' di Roma "La Sapienza", Piazzale Aldo Moro 5, I-00185, Roma - ITALY
email: `Giovanni.Dalessandro@caspur.it`
This research has been partially supported by ENEA

[3]System Assurance, ALSTOM TRANSPORT SpA, Via di Corticella 75, I-40128 Bologna - ITALY
email: `roberto.semprini@transport.alstom.com`
url: `http://www.alstom.com`
This research has been partially supported by ESPRIT project ISA-EUNET EP27450

[4]Contact Author.
Area Informatica, Università di L'Aquila, Coppito I-67100 L'Aquila - ITALY
email: `tronci@univaq.it`
url: `http://univaq.it/~tronci`
Tel: +39 0862 433129; Fax: +39 0862 433180.
This research has been partially supported by MURST project TOSCA and by ENEA.

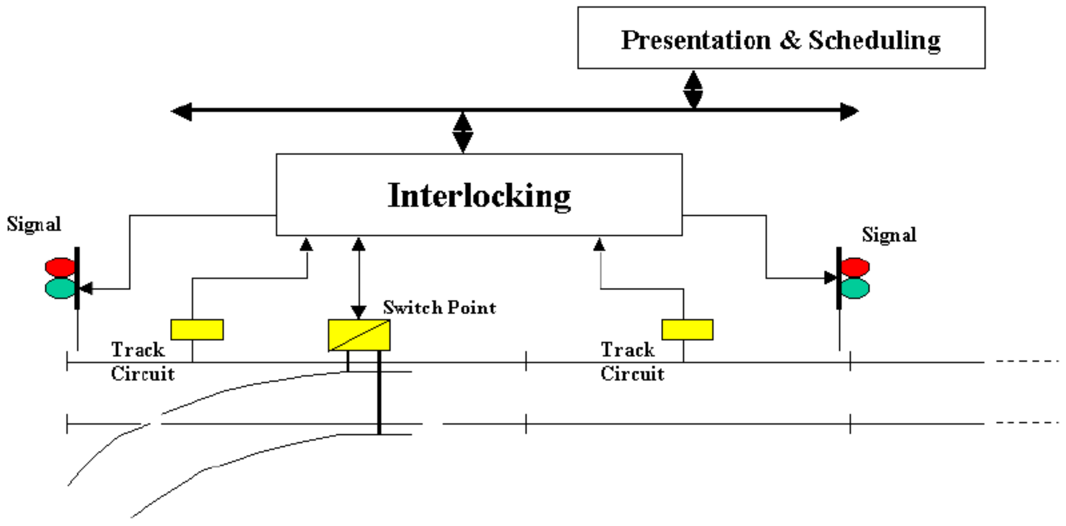

Figure 1: Interloc king System

guarantees, b y exhaustive state space exploration, that a given property holds for the system model under consideration. This increases confidence in the design correctness.

In our context a RIS implementation is defined using a finite state programming language with syn tax and semantics v ery similar to those of VCL [6], a programming language very used in RIS design.

Thus, as far as we are concerned, a RIS implementation is defined by a finite state program. Our goal here is to v erify that such program meets the giv en specifications. Since w e are applying v erification directly to the RIS program we do not have any modeling activit y. Thus w e do not ha ve modeling (abstraction) errors. In this respect we hav e the same situation that one has when verifying hardware circuits starting from e.g netlists.

The properties (safety requirements) to be verified are given to us by the signalling expert. Our task here is only that of checking that such properties hold for the RIS at hand. Thus, e.g., issues suc h as *requir ement validation* (i.e. the question "are w e asking the right thing?"), correctness of the requirement formalization process (done by hand by the signalling expert) or completeness of the set of the given requirements are not within the scope of our work. These issues are already inv estigate by the signalling experts with other approaches.

The inputs to our work are a program P defining a RIS and a formal safety requirement φ. The output of our work is the answer `YES` when P does satisfy φ or a counterexample when P does not satisfy φ.

Although model chec king is an exhaustive approach, in our context it is to be regard as a tool (among others) to *increase confidence* in the RIS design at hand. In fact, even after successful verification with a model chec ker many correctness issues may remain open. For example, the correctness of model checkers usually is not formally prov ed, it relies on testing. The same usually holds for the correctness of the softw are typically needed to interface the chosen model chec ker to the production flow.

It is not our goal to deal with the above issues. That is our aim is not *pr oving correctness* (whatever that may mean) of a given RIS. We hav e the less ambitious (as w ell as less expensive) goal of integrating, in a cost effective way, model checking in the given design flow so as to *increase confidence* in the RIS design at hand. This means that w e accept to use tools (e.g. model chec kers, interface soft w are) that are not formally verified. Note how ev er that model chec kers are widely used tools. Thus they hav e been thoroughly tested independently b y many users. Moreover interface softw are is quite simple. It typically consists of translators from a format to another. Thus typically testing suffices to find errors.

The core of all model check ers (e.g. SMV [10], Murφ [27], SPIN [28]) is the *reachability analysis*, i.e. the computation of the set of all states reachable from the given initial states. Reachability analysis can be implemented in many ways. Each of which is effective on some particular class of systems.

Order ed Binary De cision Diagrams (OBDDs) [3] have been successfully used to implement model checkers for digital circuits (e.g. [2, 10]). OBDDs provide a compact representation for boolean functions which, in turn, are used to represent the transition relation of the system to be verified as well as the set of reachable states. However the OBDD based approach to verification typically successful with hardware design is often inefficient when used for RISs because of the large number of boolean variables (easily order of thousands) occurring in RIS designs.

A typical (and often successful) approach to RIS automatic verification is based on the use of SAT solvers (i.e. tools that solve the *SATisfiability* problem for boolean functions) or tautology checkers (i.e.tools that check if a boolean function is identically 1). These approaches are, e.g., in [6, 9].

When using SAT solvers the verification problem is transformed into a satisfiability problem for boolean functions. This is done by computing the set of k-reachable states, i.e. the set of states reachable in k-steps from the initial states. This means that when we use SAT based model checking our *horizon* is bounded (by k) whereas when using OBDDs we compute the full set of states reachable from the initial states. For this reason SAT based model checking is also called *Bounded Model Checking*.

Although less expressive than (OBDD based) model checking, bounded (i.e. SAT based) model checking usually suffices to handle the typical safety requirements occurring in the RIS domain. In fact in RIS design one is typically interested in checking invariants. That is properties that are preserved during any system transition. Invariants can be checked by taking the set of all system states as the set of initial states and by computing the set of 1-reachable states. This problem can be easily transformed into a satisfiability problem for boolean functions.

One may wonder why SAT based model checking typically outperforms OBDD based model checking in the RIS domain. Intuitively this is because RISs (unlike digital circuits) are built out of weakly coupled subsystems. For a discussion on this point we refer the reader to [6, 9].

In this paper we present a case study exploring the possibility of integrating automatic formal verification methods in an industry (ALSTOM in our case) design flow for RIS.

The main obstructions to be overcome in our work are:

1. Selecting a formal verification tool that is efficient enough to solve our typical verification problems.
2. Integrating the selected tool in the design flow in a cost effective way, i.e. without requiring a complete change in the already in use design flow.
3. Keeping low the cost of writing formal safety requirements. In fact safety requirements are usually given in an informal way by the customer. To use formal verification such safety requirements have to be formalized. Essentially this has to be done by hand by signalling experts. A major retraining of these people has to be avoided.

We were able to devise a successful integration strategy meeting the above constraints. To the best of our knowlodge the issue of fully integrating automatic verification in a design flow similar to ours has not been addressed so far. Here are our main results.

1. We were able to embed SAT based model checking in our target design flow. This was done by embedding the model checker BMC (*Bounded Model Checker*, [1]) in the design flow. BMC transforms a bounded model checking problem into a satisfiability problem that can then be solved using a SAT solver. To this end we used SATO [8] which is a very efficient SAT solver.
2. We were able to integrate verification into the design flow without requiring modifications in the preexistent design flow. In fact we managed to transparently interface model checking with the existing design tools. Such interfacing only requires an easy translations of formats from the netlist like format used (by the designers) to define RISs to the input format of BMC.
3. We avoided the need for retraining of personnel (namely signalling experts). This has been achieved by writing formal requirements using the same language used to define the interlocking logic. This allows signalling experts to write formal requirements in a language very familiar to them, thus avoiding retraining of personnel.

   In our case the language used to define interlocking logic is parametric w.r.t. the railway station under consideration. As a result the formalization of safety requirements is also parametric w.r.t. the railway station under consideration. It is important to note that safety requirements depends on customer general (safety) rules and not on the particular railway station under consideration. This allows us to reuse the same formal specifications for the same customer. This is a considerable saving since in RIS designs customers (Railway Companies) do not change so often.

4. To show that our integration strategy can indeed handle *real life* RIS designs w e present experimental results of its use for the v erification of some safety requirements for an Interlocking System developed for the Singapore Subw ay. The finite state program defining such RIS has more than 2000 boolean variables. Safet y requiremen ts for suc h RIS are v erified (or a counter example is found) in seconds.

2 Design Flow

We will shortly describe the part of the target design flow relevant for our case study.

The *station layout* is giv en from the customer (i.e. a railway compan y). A station la y out defines, among other things, track connections and position of signals.

T ypically station la y outs are defined from railway companies using drawings. Examples of station layouts can be found, e.g., in [29].

For further elaboration, ho w ev er, a textual represen tation is used. Thus, from the customer format, a textual representation of the station lay out is generated. Such textual representation uses Prolog facts to define the station lay out.

Interloc king rules are given (rather informally) b y the customer. Such rules are formalized, using a Prolog-lik e language, b y signalling experts.

The Prolog facts defining the station lay out as w ell as the (formalized) in terloc king rules are giv en as input to an ALSTOM proprietary expert system called ADES2 which as output produces a finite state program defining our RIS (fig. 2). More precisely ADES2 outputs a list of boolean equations defining a *Finite State Machine* (FSM) implementing the control logic for our interloc king system.

ADES2 output is then handled to other tools that generate an EPROM implementation for the FSM generated by ADES2.

Our task is to v erify that the output produced b y ADES2 satisfies giv en safet y requirements. We must also devise a reasonable way (for ALSTOM signalling experts) to define such safet y requiremen ts.

3 Integration of Formal Verification

F or us the system to be v erified is the output of ADES2. Such output, essen tially, describes a finite state machine implementing the interloc king con trol rules. Note ho w ev er that interloc king con trol rules *are not* our safet y requiremen ts.

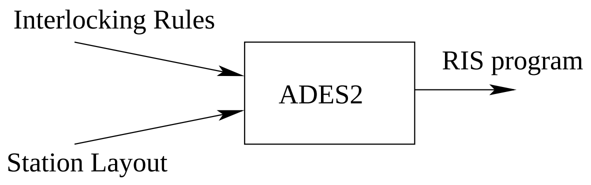

Figure 2: Our target design flow

Specifications are a list of safet y requirements described in natural language. These properties, ev entually, have to be formalized in a language suitable as input to a verification tool.

Finally, the system description and the specifications have to be fed into an automatic verification tool and the result returned to the engineers in a format familiar to them.

In the following we describe how we addressed the abo v e issues in our approac h to integration of automatic v erification in our target design flo w.

Our goal is not to explain ADES2 syntax. Thus, in order to improv e readability and sav e space in our exposition in what follows we will freely modify ADES2 syntax to make it resemble to (more or less) known languages.

3.1 F rom ADES2 format to BMC

The output of ADES2 defines the system (RIS) to be v erified. This is done by using a finite state programming language very similar to VCL [6], a programming language often used in the RIS domain. F rom this poin t of view our translation task is similar to the one in [7].

In fig. 4 is a small part of the interloc king logic (ADES2 output) for the Singapore Subw ay.

The program in fig. 4 defines a finite state sequential machine. All variables range on boolean values. Given v alues for presen t state (suffix `_NXC`) and input variables (suffix `_DI`) it defines values for next state (suffix `_CR`) and output variables (suffix `_XO`).

ADES2 programs use the following boolean operators: * (logical and), + (logical or), .N. (logical negation).

Roughly speaking semantics of ADES2 programs is as follows. In an endless loop the program is read *sequentially* from the beginning to the end. F or each equation in the program the rhs is evaluated and its value is assigned to the lhs. Of course eac h v ariable can only be defined at most once, i.e. it appears at most once on the lhs of an equation. Since equations

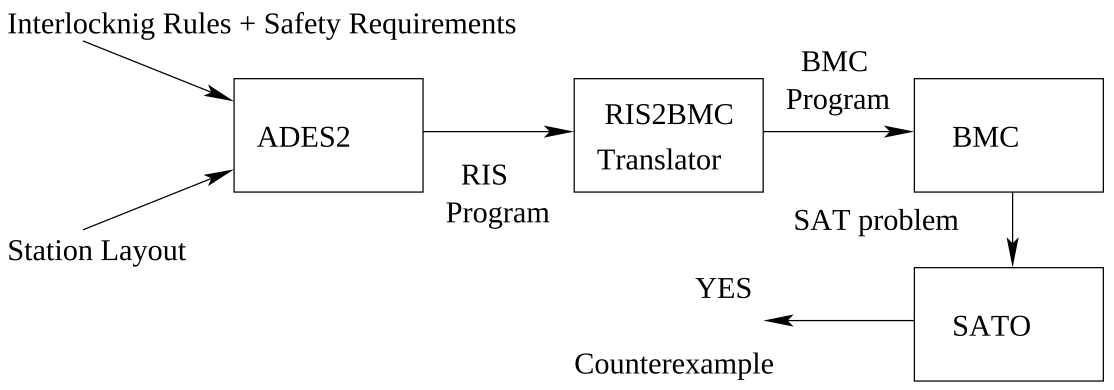

Figure 3: Target design flow integrated with automatic verification tools

are processed sequentially a state variable can only appear with suffix `_CR` on the lhs of its definition. Moreover a state variable can only appear with suffix `_NXC` in the rhs of any equation before its definition (included) and can only appear with suffix `_CR` in the rhs of any equation after its definition. These restrictions are consistent with the interpretation of suffixes `_NXC` and `_CR` as, respectively, present state and next state.

To keep our examples small in fig. 4 we have replaced most of the circuit defined by ADES2 with *fake inputs*. Such inputs are denoted with identifiers starting with the word `FAKE` in fig. 4.

Many of the properties we want to verify are *invariants*. That is properties that are supposed to hold along *any* computation path. They are typically written with a notation like $\mathbf{AG}f$ which reads as follows: "for **A**ny computation path **G**lobally f holds", where f is a propositional formula formalizing our requirement.

Automatic verification of invariants does not require the full power of model checking, *Bounded Model Checking* [1] suffices (see also section 1). This suggested us to use bounded model checking which indeed turned out to be much more effective than OBDD based model checking for our problem.

Essentially our translator takes as input an ADES2 output file and translates it into a format suitable for BMC (*Bounded Model Checker*) [1]. BMC is a bounded model checker which input language is essentially as that of SMV [10] which is a very popular OBDD based model checker.

BMC translates our bounded model checking problem into a satisfiability problem for boolean functions (*SAT problem*). In particular it translates its input into a format suitable for a certain number of SAT solvers (i.e. tools that solve a SAT problem). We used the SAT solver SATO [8].

Fig. 3 shows the design flow we obtain (from that in fig. 2) after integration with automatic verification tools.

The output of our translator for the RIS in fig. 4 is in fig. 5.

3.2 Translating Specifications

As far as we are concerned the input of ADES2 essentially consists of 2 kind of files. Files defining the layout of the railway station to be controlled and files defining the (interlocking) control rules. All together the first kind of files define a database DB.

The syntax of DB files is prolog-like, i.e. FOL (*First Order Logic*) syntax is used to define DB.

An example of (part of) the database defining the layout of the Singapore Subway is in fig. 6.

The track circuit list in fig. 6 says that identifiers of form `cdb`x (x positive integer) denote track circuits. Analogously the switch point list in fig. 6 says that identifiers of form `dev`y (y positive integer) denote switch points. In general there are many such lists defining names for each component of the railway station. The railway station layout is defined using relations. E.g. `db(is_in(dev600,cdb604))` says that the switch point `dev600` belongs to the track circuit `cdb604`.

A query on DB is essentially a FOL formula on the database. That is a query is a formula $Q(x_1, \ldots x_n)$ where $x_1, \ldots x_n$ are free variables. The result of a query is the set R of n-tuples $(X_1, \ldots X_n)$ s.t. for each $(X_1, \ldots X_n) \in R$ we have that $DB \models Q(X_1, \ldots X_n)$, i.e. $Q(X_1, \ldots X_n)$ holds in the database DB. Of course, since our database (railway station) is finite there will only be a finite set of tuples satisfying Q.

For example, using again the database in fig. 6, the

```
BOOL B604_TMR = B604_NXC
BOOL B604_CR = FAKE1_DI
BOOL B605_TMR = B605_NXC
BOOL B605_CR = FAKE2_DI
BOOL B661_TMR = B661_NXC
BOOL B661_CR = B605_CR + FAKE3_DI
BOOL IP600_CR = B604_TMR * B605_TMR
BOOL IP603_CR = B605_TMR * B661_TMR
```

Figure 4: Part of ADES2 output for the Singapore Subway Interlocking

set of tuples satisfying the formula `is_in(dev`x`, cdb`y`)` is $\{$(`dev600`, `cdb604`), (`dev603`, `cdb605`)$\}$.

Interlocking control logic is defined using rules that tell ADES2 which equations should appear in the output file. Essentially this is done by asking ADES2 to generate an equation for each tuple satisfying a given formula.

We use a C-like syntax when showing ADES2 rules generating interlocking logic.

For example the rule in fig. 7 generates the first four equations of the ADES2 output file in fig. 4.

Alstom signalling experts are familiar with ADES2 rule language since it is used to define interlocking rules (e.g. as in fig. 7). For this reason we decided to use the same language to define formal specifications too.

First of all it is to be understood that a safety requirement must hold for all railway components to which it applies. A typical safety requirement may look like:

For all track circuits `cdb`x*, for all switch points* `dev`y *s.t. switch point* `dev`y *is in track circuit* `cdb`x *we have: if* `cdb`x *is occupied by a train then* `dev`y *is blocked (i.e. cannot be moved).*

In other words, a safety requirement defines a safety property that applies to tuples (pair (`cdb`x, `dev`y) in the above case) of railway components (e.g. track circuits, switch points, signals, etc).

The set of tuples to which the safety requirement applies can be defined with a FOL formula which predicate symbols are defined using the database in input to ADES2.

In our case a generic safety requirement will have the form:

$$\forall p_1, \ldots p_n \text{ s.t. } Q(p_1, \ldots p_n) \ \mathbf{AG}F(p_1, \ldots p_n),$$

where Q is a FOL formula defining the set of tuples $(P_1, \ldots P_n)$ to which the safety requirement applies and $F(p_1, \ldots p_n)$ denotes a boolean expression defining the safety property itself.

Now, let A be the set of tuples satisfying Q. That is, $A = \{(P_1, \ldots P_n) \mid DB \models Q(P_1, \ldots P_n)\} = \{(P_{1,1}, \ldots P_{1,n}), \ldots (P_{k,1}, \ldots P_{k,n})\ \}$.

All we have to do is to check validity of the following k safety formulas: $\mathbf{AG}F(P_{1,1}, \ldots P_{1,n})$, $\ldots \mathbf{AG}F(P_{k,1}, \ldots P_{k,n})$.

Our goal is to write the above safety requirements as rules for ADES2. This is done by writing a rule that asks ADES2 to print, for each tuple $(P_{i,1}, \ldots P_{i,n}) \in A$ an equation with rhs $F(P_{i,1}, \ldots P_{i,n})$ and lhs `SPEC_`<*requirement_id*>`_`*i*`_X0`.

Such equations will be added at the end of the usual ADES2 output file (i.e. the file defining our RIS). Also buffer variables must be added to store present values `_NXC` of state variables. In fact, because of the VCL-like semantics of the programming language used to define RISs, at the end of the ouput file of ADES2 only next state `_CR` values are available. We use the prefix `OLD_` for such new buffer variables.

ADES2 can be configured so that it adds automatically the equations for such buffer (`OLD`) variables at the beginning of the output file and the equations for the `SPEC` variables at the end of the output file. We devised a way to automatically generate such configuration files for ADES2 starting from the station definition. This makes this process fully automatic.

The only thing that has to be done by hand is the formalization of the safety requirement. This can be done using the rule language of ADES2. It is important to note that this formalization step has to be done by a signalling expert since very specific domain knowledge is needed. Thus using a specification language (namely ADES2 rule language) known to signalling experts is crucial to make our integration useful.

An example will hopefully clarify the matter. Consider the (part of) database in fig. 6. Suppose we want to verify the following requirement:

SR1: *If the Track Circuit is occupied then Point Locking Status shall not be cleared.*

```
DEFINE
B604_TMR := B604_NXC ;
B604_CR := FAKE1_DI ;
B605_TMR := B605_NXC ;
B605_CR := FAKE2_DI ;
B661_TMR := B661_NXC ;
B661_CR := B605_CR | FAKE3_DI ;
IP600_CR := B604_TMR & B605_TMR ;
IP603_CR := B605_TMR & B661_TMR ;


ASSIGN
next(B604_NXC) := B604_CR;
next(B605_NXC) := B605_CR;
next(B661_NXC) := B661_CR;
next(IP600_NXC) := IP600_CR;
next(IP603_NXC) := IP603_CR;

-- Translation of ADES2 output ends here.

-- Spec1
--SPEC AG(!B604_NXC ->  !IP600_CR)

-- Spec2
SPEC AG(!B605_NXC ->  !IP603_CR)
```

Figure 5: BMC translation of ADES2 output file in fig. 4

```
/***  TRACK CIRCUITS   ***/
db(cdb(cdb604)).
db(cdb(cdb605)).

/* SWITCH POINTS */
db(switch_point(dev600)).
db(switch_point(dev601)).
db(switch_point(dev602)).
db(switch_point(dev603)).

/* SWITCH POINTS, TRACK CIRCUITS */
db(is_in(dev600,cdb604)).
db(is_in(dev603,cdb605)).
```

Figure 6: Example of database in input to ADES2

```
i = 1; forall cdbx s.t.  db(cdb(cdbx)) {
            printf(``B%d_TMR = B%d_NXC\n'', x);
            printf(``B%d_CR = FAKE%d_DI\n'', x, i); i++; }
```

Figure 7: Example of rules input to ADES2

```
i = 1;
forall (cdbx, devy) s.t.  db(is_in(devy, cdbx)) {
      printf(``SPEC_SR1_%d_XO = OLD_B_%d_XO + .N.IP_%d_CR\n'', i, x, y);
      i++; }
```

Figure 8: Formalization of safety requirements as ADES2 rules

```
BOOL OLD_B604_XO = B604_NXC
BOOL OLD_B605_XO = B605_NXC
BOOL OLD_B661_XO = B661_NXC
BOOL OLD_IP600_XO = IP600_NXC
BOOL OLD_IP603_XO = IP603_NXC

BOOL B604_TMR = B604_NXC
BOOL B604_CR = FAKE1_DI
BOOL B605_TMR = B605_NXC
BOOL B605_CR = FAKE2_DI
BOOL B661_TMR = B661_NXC
BOOL B661_CR = B605_CR + FAKE3_DI
BOOL IP600_CR = B604_TMR * B605_TMR
BOOL IP603_CR = B605_TMR * B661_TMR

BOOL SPEC_SR1_1_XO = OLD_B604_XO + .N.IP600_CR
BOOL SPEC_SR1_2_XO = OLD_B605_XO + .N.IP603_CR
```

Figure 9: ADES2 output using safety rule in fig. 8

Here is how our signalling expert formalized such rule for us.

Step 1. Rule SR1 means the following. For all track circuits cdb$x$, for all switch points dev$y$ s.t. the switch point dev$y$ is in the track circuit cdb$x$ the following must hold: if cdb$x$ is occupied by a train then dev$y$ is blocked.

Step 2 Trackcircuit cdb$x$ is occupied iff variable B_x_NXC is 0 (false). The switch point is blocked when variable IP_y is 0 (false).

Step 3. A possible ADES2 rule formalizing safety requirement SR1 is in fig. 8.

Note how step1 and, even more, step 2, require a specific signalling expertise.

The output of ADES2 when the rule in fig. 8 is added to the other rules is in fig. 9.

It is important to relize that the rule in fig. 8 formalizing the safety requirement SR1 does not change when we change the railway station layout. That is it is parametric w.r.t. the station layout. Of course a finite state model checker cannot handle requirements that are parametric w.r.t. the station layout. ADES2 takes care of instatiating automatically the safety requirements accordingly to the station layout. E.g. the equations produced automatically by ADES2 when using the rule in fig 8 are in fig. 9. As a result, as far as the signalling expert is concerned, formal safety requirements change only when informal safety requirements change. That is when the customer (Railway Company) changes, which does not happen very often.

This is a very important feature of our integration strategy since, to some extent, allows the signalling expert to work "the usual way".

Our translator from ADES2 output file to BMC recognizes SPEC variables (i.e. identifiers starting with the word SPEC) and outputs the file in fig. 10 when given in input the file in fig. 9.

Note that BMC only handles one specification (introduced by the keyword SPEC) at a time. Thus to verify n SPECs it must be run n times by commenting out all but one SPEC. This can, of course, be done automatically. Since verifying a requirement implies checking many formulas it is important that verification of a single formula can be done in a short time. In our case it is a matter of fractions of seconds.

4 Experimental results

Since the system to be verified (output from ADES2) and the specifications (defined via ADES2 rules) are

```
DEFINE
OLD_B604_XO := B604_NXC ;
OLD_B605_XO := B605_NXC ;
OLD_B661_XO := B661_NXC ;
OLD_IP600_XO := IP600_NXC ;
OLD_IP603_XO := IP603_NXC ;

B604_TMR := B604_NXC ;
B604_CR := FAKE1_DI ;
B605_TMR := B605_NXC ;
B605_CR := FAKE2_DI ;
B661_TMR := B661_NXC ;
B661_CR := B605_CR | FAKE3_DI ;
IP600_CR := B604_TMR & B605_TMR ;
IP603_CR := B605_TMR & B661_TMR ;

SPEC_SR1_1_XO := OLD_B604_XO | !IP600_CR ;
SPEC_SR1_2_XO := OLD_B605_XO | !IP603_CR ;

ASSIGN
next(B604_NXC) := B604_CR;
next(B605_NXC) := B605_CR;
next(B661_NXC) := B661_CR;
next(IP600_NXC) := IP600_CR;
next(IP603_NXC) := IP603_CR;

-- Translation of ADES2 output ends here.

-- Spec1
-- SPEC AG SPEC_SR1_1_XO

-- Spec2
SPEC AG SPEC_SR1_2_XO
```

Figure 10: BMC translation of ADES2 file in fig. 9

Requirement ID	Instance of formal Req
SR1	$\neg \texttt{OLD\_B605\_XO} \rightarrow \neg \texttt{IP603\_CR}$
SR2	$(\neg \texttt{DPR603\_DI} \wedge \neg \texttt{OLD\_R\_CPPR603\_XO} \wedge \neg \texttt{OLD\_PPN603\_XO} \wedge \neg \texttt{VPL603\_CR}) \rightarrow (\texttt{OLD\_PPR603\_XO} = \texttt{PPR603\_CR})$
SR3	$(\neg \texttt{DPR603\_DI} \wedge \neg \texttt{OLD\_R\_CPPR603\_XO} \wedge \neg \texttt{OLD\_PPN603\_XO} \wedge \neg \texttt{IP603\_CR}) \rightarrow (\texttt{OLD\_PPR603\_XO} = \texttt{PPR603\_CR})$
SR4	$(\neg \texttt{DPN603\_DI} \wedge \neg \texttt{OLD\_R\_CPPR603\_XO} \wedge \neg \texttt{OLD\_PPR603\_XO} \wedge \neg \texttt{VPL603\_CR}) \rightarrow (\texttt{OLD\_PPN603\_XO} = \texttt{PPN603\_CR})$
SR5	$(\neg \texttt{DPN603\_DI} \wedge \neg \texttt{OLD\_R\_CPPR603\_XO} \wedge \neg \texttt{OLD\_PPR603\_XO} \wedge \neg \texttt{IP603\_CR}) \rightarrow (\texttt{OLD\_PPN603\_XO} = \texttt{PPN603\_CR})$

Figure 11: Some Safety Requirement

Req Id	SR1	SR2	SR3	SR4	SR5
CPU (sec)	0.23	0.27	0.29	0.34	0.34

Figure 12: V erification Times on a 200 MHz Pentium Linux PC with 112 MB of RAM

defined using *First Order Logic* (FOL) we decided, as a first step, to use BSP (*Boolean Symbolic Programming*) [11, 12] to carry out our experiments. BSP is, essentially, a FOL based interface to OBDD packages. We used BSP to drive the *Colorado University Decision Diagram* (CUDD) package [13]. This approach allowed us to quickly carry out some experiment, since almost no interfacing work had to be done between ADES2 output and the model checker at hand (BSP).

Unfortunately the interlocking system for an average size station can easily require 1000 or more state variables. Verification times even for simple properties were prohibitive using OBDDs.

This suggested us to use the approach described in this paper. Namely using SAT based model checking via BMC [1] and SATO [8].

In the following we show some of the experiments we run to assess effectiveness of the approach presented in this paper. The goal of such experiments is to show that we can handle the typical RIS verification problem arising in our given industrial environment. Our experiments show that indeed a given safety requirement can be automatically verified (or a counter example can be generated) in fraction of seconds.

Table 11 gives some of the properties we verified together with an instantiation of the expression formalizing the property. The first column in fig. 11 gives the name (for further reference) of the safety requirement whereas the second column gives an instance of the requirement for a particular set of station devices. The second column in fig. 11 is shown just to give the reader an idea of how a formalized requirement looks like in our case. The identifiers in the second column of fig. 11 that are not in fig. 9 are in parts of the RIS circuit that we have not reported in our present exposition.

Table 12 gives CPU time to verify the properties shown in table 11. The first row of fig. 12 gives the safety requirement identifier (defined in the first column of fig. 11) whereas the second row in fig. 12 gives CPU times in seconds. Memory usage was alway about 3MB.

5 Conclusions

We presented a case study aimed at exploring the possibility of integrating automatic formal verification methods in an industry design flow for *Railway Interlocking Systems* (RIS).

The main difficulties we had to overcome were: selecting a formal verification tool efficient enough to solve the verification problems at hand and devising a cost effective integration strategy for such tool.

We were able to effectively integrate SAT based *Bounded Model Checking* in our target design flow. We achieved this without requiring modifications in the design flow nor retraining of personnel. Moreover we devised an architecture that allows for reuse of formal specifications across different railway stations as long as the informal set of requirements does not change (i.e. as long as the RIS customer does not change).

To asses effectiveness of our approach on *real life* RIS designs we have verified some safety properties of a Railway Interlocking System designed for the Singapore Subway. Such RIS has more than 2000 boolean variables. Verification of each property took less than 0.5 seconds on our machine.

References

[1] A. Biere, A. Cimatti, E. Clarke, Y. Zhu, *Simbolic Model Checking Without BDDs*, Proceedings of TACAS 1999, LNCS 1579, Springer, 1999

[2] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, *Symbolic model checking: 10^{20} states and beyond*, Information and Computation 98, (1992)

[3] R. Bryant, *Graph-Based Algorithms for Boolean Function Manipulation*, IEEE Trans. on C., Vol. C-35, N.8, Aug. 1986

[4] CEN/CENELEC, prEN50128, *Railway Applications: Software for Railway Control and Protection Systems.*

[5] CEN/CENELEC, prEN50129, *Railway Applications: Safety Related Electronic Railway Control and Protection Systems.*

[6] J. F. Groote, J. W. C. Koorn, S.F.M. van Vlijmen, *The Safety Guaranteeing System at Station Hoorn-Kersenboogerd*, Technical Report 121, Logic Group Preprint Series, Utrecht University, 1994

[7] Cindy Eisner, *Using Symbolic Model Checking to Verify the Railway Stations of Hoorn-Kersenboogerd and Heerhugowaard*, Proceedings of CHARME 1999, LNCS 1703, Springer, 1999

[8] H. Zhang, *SATO: An Efficient Propositional Prover*, Proceedings of "International Conference on Automated Deduction" (CADE'97), LNAI 1249, Springer-Verlag, 1997

[9] M. Sheeran, G. Stalmarck, *A Tutorial on Stalmarck's Proof Procedure for Propositional Logic*, Second International Conference on "Formal Methods in Computer Aieded Design", FMCAD'98, Vol. 1522 of LNCS, 1998, Springer

[10] K.L. McMillan, *Symbolic Model Checking: An Approach to the State Explosion Problem*, Kluwer Academic Publishers, 1993

[11] E. Tronci, *Hardware Verification, Boolean Logic Programming, Boolean Functional Programming*, Proc. 10th IEEE Conf. on "Logic In Computer Science" 1995, San Diego, CA, USA

[12] `http://univaq.it/~tronci/bsp.html`

[13] F. Somenzi, *CUDD, CU Decision Diagram Package*, ftp://vlsi.colorado.edu/pub/

[14] A.Anselmi, C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, F. Torielli, *An Experience in Formal Verification of Safety Properties of a Railway Signalling Control System*, Proc. SAFECOMP '95, Springer - Verlag, 1995.

[15] C. Bernardeschi, A. Fantechi, S. Gnesi, G. Mongardi, *Proving safety properties for embedded control systems*, Proc. EDCC-2, LNCS 1150, Springer - Verlag, 1996.

[16] C. Bernardeschi, A. Fantechi, S. Gnesi, G. Mongardi, *Formal verification of safety requirements on complex systems*, Proc. SAFECOMP '96, Springer - Verlag, 1996

[17] C. Bernardeschi, A. Fantechi, S. Gnesi, S. Larosa, G. Mongardi, D. Romano, *Formal Verification Environment for Railway Signalling System Design*, Formal Methods in System Design, Vol 12, n. 2, Kluwer Academic Publishers, 1998

[18] Arne Borälv, *A Fully Automated Approach for Proving Safety Properties in Interlocking Software Using Automatic Theorem-Proving*, Proceedings of the Second International ERCIM Workshop on Formal Methods for Industrial Critical Systems, pp. 39-62, Consiglio Nazionale Ricerche, Pisa, July 1997

[19] B. Dehbonei, F. Mejia, *Formal Development of software in railways safety critical systems*, Railway Operations, Vol. 2, pp. 213-219, Computational Mechanics Publications, 1994.

[20] W. J. Fokkink, *Safety criteria for the vital processor interlocking at Hoorn–Kersenboogerd*, Proceedings of the 5th Conference on Computers in Railways, COMPRAIL'96, Part I: Railway Systems and Management, pp. 101-110, Computational Mechanics Publications, 1996.

[21] W. J. Fokkink, G. P. Kolk, S. F. M. van Vlijmen, *EURIS, a specification method for distributed interlockings*, Proceedings of SAFECOMP '98, LNCS, Springer-Verlag, 1998.

[22] Takahiko OGINO, Mitsuyoshi FUKUDA, Yuji HIRAO, *VDM Specification of an Interlocking System and a Simulator for its Validation*, WCRR (World Conference on Railway Research), 1996

[23] Kirsten Mark Hansen, *Formalising Railway Interlocking Systems*, Nordic Seminar on Dependable Computing Systems, pp. 83-94, Department of Computer Science, August 1994.

[24] Kirsten Mark Hansen, *Validation of a Railway Interlocking Model*, FME'94: Industrial Benefit of Formal Methods, pp. 582-601, Springer-Verlag, October 1994.

[25] M. J. Morley, *Safety-level Communication in Railway Interlockings*, Science of Computer Programming, Vol. 29, Number 1-2, pp. 147-170, July 1997.

[26] Marten Säflund, *Modelling and Formally Verifying Systems and Software in Industrial Applications*, Proceedings of the second International Conference on "Reliability, Maintainability and Safety (ICRMS '94) Beijing", pp. 169-174, International Academic Publishers, June 1994.

[27] D. L. Dill and A. J. Drexler and A. J. Hu and C. H. Yang, *Protocol Verification as a Hardware Design Aid*, IEEE International Conference on Computer Design: VLSI in Computers and Processors, 1992

[28] G. J. Holzmann, *The Spin Model Checker*, IEEE Trans. on Software Engineering, Vol. 23, N. 5, May 1997, pp. 279–295

[29] url: `http://www.trainweb.com`